

CS526 O2
Homework Assignment 2

Problem 1 (20 points). This is practice for analyzing running time of algorithms. Express the running time of the following methods, which are written in pseudocode, using the *big-oh* notation. Assume that all variables are appropriately declared. You must justify your answers. If you show only answers, you will not get any credit even if they are correct.

(1)

```
method1(int[ ] a) // returns integer
x = 0;           //c1
y = 0;           //c2
for (i=1; i<n; i++) { //n-1 // n is the number of elements in array a
    if (a[i] == a[i-1]) { //c3
        x = x + 1;
    }
    else {
        y = y + 1;
    }
}
return (x - y);    //c4
```

Answer: x takes c1 constant time, y takes c2 constant time, for loop and loop body of the for loop runs n-1 linear times, loop body c3 takes constant time, return statement takes c4 constant time.

We do not count constant time here and we only take the longest runtime with the worst scenario.

Total running time:

$$f(n) = c1 + c2 + c3(n-1) + c4$$

$$f(n) = c3(n-1)$$

$$f(n) = c3n - c3$$

$$\underline{F(n) = O(n)}$$

(2)

```
method2(int[ ] a, int[ ] b) // assume equal-length arrays
    x = 0;           //c1
    i = 0;           //c2
    while (i < n) { //c3(n-1) // n is the number of elements in each array
        y = 0;       //c4
        j = 0;       //c5
        while (j < n) { //c6(n-1)
            k = 0;     //c7
            while (k <= j) { //c8(n)
                y = y + a[k]; //c9(1)
                k = k + 1;    //c10(1)
            }
            j = j + 1; //c11(1)
        }
    }
```

```

    }
    if (b[i] == y) {    // c12(1)
        x++;
    }
    i = i +
    1;
    //c13(1)
}
return x;    //c14(1)

```

Answer: x takes c_1 constant time, i takes c_2 constant time, while loop ($i < n$) and loop body for the while loop runs $c_3(n-1)$ linear times, loop body y takes c_4 constant time, j takes c_5 constant time, while loop ($j < n$) runs $c_6(n-1)$ linear times, loop body k takes c_7 constant time, while loop ($k \leq j$) takes $c_8(n)$ linear time, loop body c_9 , c_{10} , c_{11} takes constant time, if statement ($b[i] == y$) runs $c_{12}(1)$ linear time with $b[i] == y$ max $n-1$ time, if statement body c_{13} linear time, return statement takes c_{14} constant time.

We do not count constant time here and we only take the longest runtime with the worst scenario.

Total running time:

$$f(n) = (c_1 + c_2 + c_3(n-1) + c_{14}) * (c_4 + c_5 + c_6(n-1) + c_{12} + c_{13}) * (c_7 + c_8(n) + c_{11}) * (c_9 + c_{10})$$

$$f(n) = c_3(n-1) * c_6(n-1) * c_8(n) = n * n * n$$

$$f(n) = n^3$$

$F(n) = O(n^3)$

(3)

```

// n is the length of array a
// p is an array of integers of length 2
// initial call: method3(a, n-1, p)
// initially p[0] = 0, p[1] = 0
method3(int[] a, int i, int[] p)
    if (i == 0) {    //c1 - O(1)
        p[0] = a[0];
        p[1] = a[0];
    }
    else {
        method3(a, i-1, p);    //c2(n-1) - O(n-1)
        if (a[i] < p[0]) {    //c3 - O(1)
            p[0] = a[i];
        }
        if (a[i] > p[1]) {    // c4 - O(1)
            p[1] = a[i];
        }
    }
}

```

Answer: if else statement ($i == 0$) with its body runs $c1$ constant time, recursive method3 runs $c2(n-1)$ linear time as the input size is reduced 1 with each recursive call, this function is being called recursively n times before reaching the base case. If statement ($a[i] < p[0]$) runs $c3$ constant time, if statement ($a[i] > p[1]$) runs $c4$ constant time.

We do not count constant time here and we only take the longest runtime with the worst scenario.

Total running time:

$$f(n) = c1 + c2(n-1) + c3 + c4$$

$$f(n) = c2(n-1)$$

$$f(n) = c2n - c2$$

$$f(n) = n$$

$$\underline{F(n) = O(n)}$$

(4)

```
// initial call: method4(a, 0, n-1) // n is the length of array a
public static int method4(int[] a, int x, int y)
{
    if (x >= y) //c1 - O(1)
    {
        return a[x];
    }
    else
    {
        z = (x + y) / 2; // c2 - O(1) - integer division
        u = method4(a, x, z); //c3(n) - O(n)
        v = method4(a, z+1, y); //c4(n+1) - O(n)
                                //O(2^n)
        if (u < v) return u;    //c5 - O(1)
        else return v;
    }
}
```

Answer: if else statement ($x \geq y$) with its body runs $c1$ constant time and it splits into two possible sequences, in the else statement where z variable takes $c2$ constant time to get the integer division. The variable u takes a recursive call which runs $c3$ linear time since the function is being called recursively n times before reaching the base case, and the variable v takes a recursive call which runs $c4$ linear time. With $c3$ in the same block, it's $O(2^n)$ exponential times, since each function call calls itself twice unless it has been recursed n times, and v has to wait until u resolves its recursive call and unwind before v starts its recursive call. The if else statement ($u < v$) with its body runs $c5$ constant time. We do not count constant time here and we only take the longest runtime with the worst scenario.

Total running time:

$$f(n) = c1 + c2 + (2^n) + c5$$

$$\underline{F(n) = O(2^n)}$$

The running time is $O(n)$.
We are recursing on each half.

Problem 2 (20 points) This problem is about the stack and the queue data structures that are described in the textbook.

(1) Suppose that you execute the following sequence of operations on an initially empty stack. Using Example 6.3 in the textbook as a model, complete the following table.

Operation	Return Value	Stack Contents
push(10)		(10)
pop()	10	()
push(12)		(12)
push(20)		(12, 20)
size()	2	(12, 20)
push(7)		(12, 20, 7)
pop()	7	(12, 20)
top()	20	(12, 20)
pop()	20	(12)
pop()	12	()
push(35)		(35)
isEmpty()	false	(35)

(2) Suppose that you execute the following sequence of operations on an initially empty queue. Using Example 6.4 in the textbook as a model, complete the following table.

Operation	Return Value	Queue Contents (first \leftarrow Q \leftarrow last)
enqueue(7)		(7)
dequeue()	7	()
enqueue(15)		(15)
enqueue(3)		(15, 3)
first()	15	(15, 3)
dequeue()	15	(3)
dequeue()	3	()
first()	null	()
enqueue(11)		(11)
dequeue()	11	()
isEmpty()	true	()